applied sciences

MDPI

Article

Containerized Microservices Orchestration and Provisioning in Cloud Computing: A Conceptual Framework and Future Perspectives

Abdul Saboor [1,*], Mohd Fadzil Hassan [2], Rehan Akbar [3], Syed Nasir Mehmood Shah [4], Farrukh Hassan [1], Saeed Ahmed Magsi [5] and Muhammad Aadil Siddiqui [5]

[1] High-Performance Cloud Computing Centre (HPC3), Department of Computer & Information Sciences, Universiti Teknologi PETRONAS, Seri Iskandar 32610, Perak Darul Ridzuan, Malaysia; farrukh_18001246@utp.edu.my
[2] Centre for Research in Data Science (CeRDaS), Department of Computer & Information Sciences, Universiti Teknologi PETRONAS, Seri Iskandar 32610, Perak Darul Ridzuan, Malaysia; mfadzil_hassan@utp.edu.my
[3] Positive Computing (+COMP), Department of Computer & Information Sciences, Universiti Teknologi PETRONAS, Seri Iskandar 32610, Perak Darul Ridzuan, Malaysia; rehan.akbar@utp.edu.my
[4] KICSIT, Institute of Space Technology (IST), Islamabad 44000, Pakistan; nasirsyed.utp@gmail.com
[5] Department of Electrical and Electronics Engineering, Universiti Teknologi PETRONAS, Seri Iskandar 32610, Perak Darul Ridzuan, Malaysia; saeed_19001716@utp.edu.my (S.A.M.); muhammad_18003606@utp.edu.my (M.A.S.)
* Correspondence: abdul_19001745@utp.edu.my

Abstract: Cloud computing is a rapidly growing paradigm which has evolved from having a monolithic to microservices architecture. The importance of cloud data centers has expanded dramatically in the previous decade, and they are now regarded as the backbone of the modern economy. Cloud-based microservices architecture is incorporated by firms such as Netflix, Twitter, eBay, Amazon, Hailo, Groupon, and Zalando. Such cloud computing arrangements deal with the parallel deployment of data-intensive workloads in real time. Moreover, commonly utilized cloud services such as the web and email require continuous operation without interruption. For that purpose, cloud service providers must optimize resource management, efficient energy usage, and carbon footprint reduction. This study presents a conceptual framework to manage the high amount of microservice execution while reducing response time, energy consumption, and execution costs. The proposed framework suggests four key agent services: (1) intelligent partitioning: responsible for microservice classification; (2) dynamic allocation: used for pre-execution distribution of microservices among containers and then makes decisions for dynamic allocation of microservices at runtime; (3) resource optimization: in charge of shifting workloads and ensuring optimal resource use; (4) mutation actions: these are based on procedures that will mutate the microservices based on cloud data center workloads. The suggested framework was partially evaluated using a custom-built simulation environment, which demonstrated its efficiency and potential for implementation in a cloud computing context. The findings show that the engrossment of suggested services can lead to a reduced number of network calls, lower energy consumption, and relatively reduced carbon dioxide emissions.

Keywords: cloud computing; virtual machine; containers; microservices; multicloud

Citation: Saboor, A.; Hassan, M.F.; Akbar, R.; Shah, S.N.M.; Hassan, F.; Magsi, S.A.; Siddiqui, M.A. Containerized Microservices Orchestration and Provisioning in Cloud Computing: A Conceptual Framework and Future Perspectives. *Appl. Sci.* **2022**, *12*, 5793. https://doi.org/10.3390/app12125793

Academic Editor: Eui-Nam Huh

Received: 31 March 2022
Accepted: 19 April 2022
Published: 7 June 2022

1. Introduction

In recent years, cloud computing has gained the interest of industries and of researchers. Cloud computing allows ubiquitous computing and delivers the proper on-demand admittance to the common pool of configurable resources, such as memory storage, network, compute nodes, and applications [1]. Popular cloud service providers which provide such services are Microsoft Azure, Amazon Web Services (AWS), Google Cloud

Platform (GCP), IBM Cloud, and Kamatera. Elasticity, the pay-as-you-go model, and on-demand provisioning of resources are a few of the major cloud computing advantages [2]. Three major categories of the cloud service models are known as [1,3,4] Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS); refer to Figure 1.

(1) Software as a Service (SaaS): SaaS provides access to cloud-based software. Through web or API, the users access the software residing on the remote cloud network. In SaaS, the vendor manages most of the technical issues including data handling, storage, servers, etc. Therefore, the users do not have to spend more time installing, maintaining, and upgrading software. Some of the SaaS examples are Dropbox, Salesforce, GoToMeeting, Google Apps, etc.

(2) Infrastructure as a Service (IaaS): In IaaS, the cloud computing vendors provide access to resources such as storage, computing resources, and networking. IaaS lets businesses purchase resources according to their needs/demands. Therefore, IaaS provides a flexible cloud computing model where clients have complete control over the infrastructure. As the whole infrastructure is based on the cloud, there will be no single point of failure. Some of the IaaS examples are Amazon Web Services (AWS), Microsoft Azure, Google Compute Engine (GCE), etc.

(3) Platform as a Service (PaaS): PaaS provides a framework for the users (specifically developers) on which they can develop, manage, and deploy the applications. The users not only have access to computing and storage resources, but they also have access to a set of tools for application development, testing, security, backups, etc. PaaS has several advantages including scalability, high availability, and a cost-effective development and deployment platform. Some of the PaaS examples are Apache Stratos, AWS Elastic Beanstalk, Google App Engine, etc.

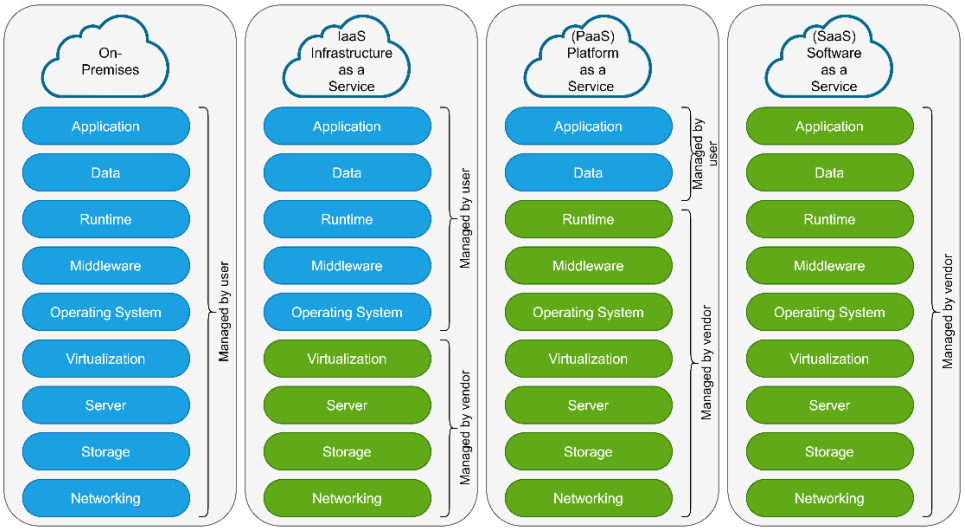

Figure 1. IaaS-PaaS-SaaS comparison [4].

In the last decade, large-scale, as well as small- to medium-sized, enterprises mostly shifted to cloud computing. As cloud computing deals with the parallel deployment of data-intensive workloads in real-time, such deployments bring in various challenges, such as optimal resource scheduling, power utilization, network latency, task scheduling, etc. While dealing with these challenges, the cloud computing environment also needs to make efficient use of energy to minimize the operational cost and introduce eco-friendly data centers. Studies have shown that cloud resources are being used inefficiently [5,6] and the continuous shift of technologies for the cloud environment requires efficient task and

resource handling while reducing energy consumption and lowering the carbon footprint produced by the hardware used in the cloud platform.

Due to the complexity of cloud services, it is appropriate to illustrate a conceptual framework through a modular and multilevel stratified architecture that enables the paradigm of containerized microservices orchestration and provisioning in cloud computing. For this study, the formulation of the conceptual framework commences with the subject of microservices orchestration and provisioning and then moved on to exhaustive literature research and began identifying the important ideas applied by previous studies. Once it determines what has been accomplished by previous researchers and what needs to be accomplished by locating a clear call-to-action described in the literature, this study begins to extract variables, concepts, theories, and current frameworks outlined in the appropriate literature. The framework development process is initiated by understanding how some of the variables, concepts, theories, and features of current frameworks interact.

For the rest of this paper, Section 2 provides an intensive literature review of cloud-based containerized services and articulates the unique challenges. Section 3 describes the methodology and elaborates on the proposed framework. To justify the working efficiency of the proposed framework, it was partially tested in a simulation environment and the results are discussed in Section 4. Finally, Section 5 concludes the study and advises future directions.

2. Literature Review

In cloud computing, one of the most critical factors is to provide quality of service (QoS) within the defined constraints. Few of the constraints include response time, throughput, minimum service unavailability, low cost, execution, or completion time limits. This section outlines some of the existing research studies performed to meet the QoS constraints.

Traditionally, heuristic scheduling algorithms have been used to minimize the processing time and schedule tasks efficiently. A study by Juarez et al. [7] used the polynomial-time algorithm that combines the set of heuristic rules and resource allocation techniques to execute task-based applications efficiently on distributed computing platforms. Recently, Qiu et al. [8] addressed the pricing policies of cloud providers to cloud service brokerages (CSBs) and found a pricing policy for cloud providers which will maximize the CSBs' profit by minimizing the cloud providers' energy costs. Another study [5] proposed a dynamic energy-aware cloudlet-based mobile cloud computing model, which focused on solving additional energy consumption during wireless communication.

Multiple other studies addressed the technical and nontechnical aspects of cloud computing, but still, many gray areas need to be addressed. Some of the areas include dynamic allocation of resources and energy, virtualization techniques for balancing the energy-aware workload between virtual machines, and thermal-aware management techniques. Horri et al. [9] proposed a novel QoS-aware VM consolidation method based on the resource utilization history of VM. The results showed improvement in QoS metrics and energy consumption. Similarly, Esfandiarpoor [10] used the concept of VM consolidation but extended it to physical machines such that fewer racks and routers were used. This resulted in turning off the idle routing and cooling equipment and the experimental results showed a significant reduction in energy consumption.

Since the previous decade, there has been a trend toward containerization technologies. The principle of containers enables a flexible and lightweight environment that allows software programs to start sharing the underlying operating system. As a result, containers are a type of operating system virtualization. A standard container may host huge application processes via microservices. Containers contrast with virtual machines where the virtual machines execute the whole guest operating system, but it is not the case with containers, as shown in Figure 2. Container technology has several industrial applications. A few of the industries where the containers have been adopted are [11]: (a) containers in IoT applications; (b) containerization in gaming; (c) containerization in healthcare; (d) containerization in web applications; (e) containers in financial services

providers; (f) containerization in microservices architectures; (g) containerization in marketing and advertising; (h) containerization in scientific workflows; (i) containerization in edge computing.

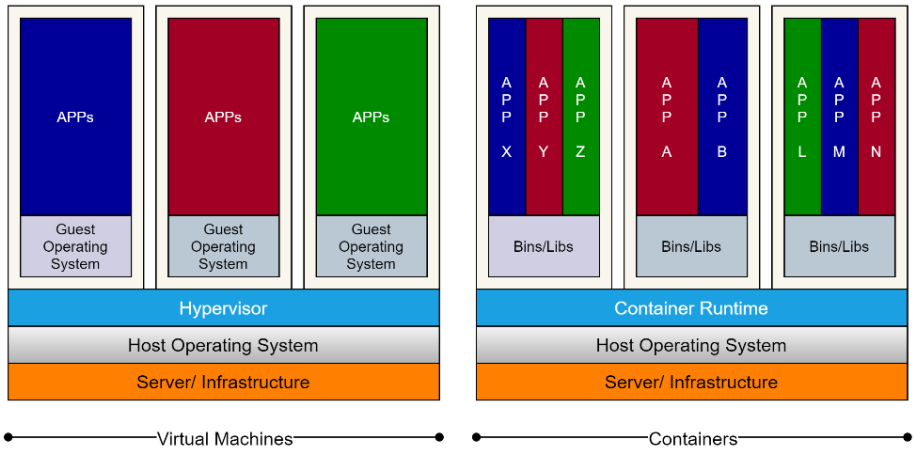

Figure 2. Difference between VM and containers.

The performance of virtual machines is stable when compared to containers, but VMs have bigger memory footprints and VMs also have slower boot-up and shut-down downtime [11]. Multiple studies addressed containerization in the cloud computing environment. The container placement strategy in a CaaS as proposed by Zhang et al. [12] achieved energy savings by optimizing. They did the placement of containers using an improved genetic algorithm (IGA). The containers were placed on VMs which were hosted on physical machines (PM). It was considered that VMs could meet the resource requirement of all new containers, and only two resources were considered, i.e., CPU and memory. According to the authors, in the conventional genetic algorithm (GA), it is hard to obtain a better chromosome (consisting of container ID and VM ID) that fits better when VM resource utilization is high. Therefore, the introduced IGA showed better results in terms of energy saving when compared to spread and bin-packing strategies and first-fit and conventional GA. However, the study only considered the energy saving parameter, and parameters such as high availability and low resource wastage were not studied. According to the authors, the proposed strategies only apply to the static placement of containers and were unable to support real-time scheduling and relocation. Thus, dynamic container consolidation is still an open issue.

Docker containers are used to perform OS-level virtualization. However, according to Zhang et al. [13], container deployment on VM is a challenge. Because VM placement and container placement need not be addressed separately, the study thus presented the Container-VM-PM approach. The method improved the deployment of new containers while reducing the number of physical machines and resources lost.

To solve the issue of significant communication demands across containers, Lv et al. [14] developed container distribution schemes. The study presented a worst-fit reducing method for container deployment and a two-stage sweep and search algorithm was introduced for container assignment. The evaluation findings demonstrated a decrease in communication overhead between containers. Xu and Buyya [6] worked for energy-efficient cloud computing and used brownout schema for software systems to dynamically activate or deactivate optional microservices. The system used Docker Swarm for container management, i.e., the activation and deactivation of the container. The study did not address memory management, resource management (including cooling system infrastructure), and network congestion issues. The experiment was performed using a custom-built web application and needs testing for more dynamic and intensive workloads.

Hussein et al. [15] improved the resource utilization of CPU cores and memory and minimized the number of newly created VMs and active PMs. The proposed ant colony optimization based on a best-fit algorithm (called metaheuristic ACO) showed significant improvement in the resource utilization of PMs and VMs. However, the energy efficiency parameter was not considered, and the time costs of the heuristic algorithms (which are computationally lightweight) were lower than that of metaheuristic ACO. Srikantaiah [16] was among the first researchers to address the problem of consolidation to improve energy efficiency in the cloud computing environment. Later, Tchana et al. [17] incorporated the consolidation of virtual machines and support real-time sizing algorithms to eliminate resource gaps (i.e., unused resources). The studies were carried out in a private cloud utilizing a VM or PM; however, just the Tomcat servlet was offered to host by the container.

Kaewkasi and Chuenmuneewong [18] used ant colony optimization (ACO) for the scheduling of Docker containers. The ACO algorithm was used to distribute application containers over the Dockers' hosts for efficient resource usage. The experiments showed that ACO performance was higher by 15% when compared to the greedy scheduler algorithm. However, the study considered monolithic application implementation.

Recently, an experimental study [11] was conducted to compare the performance of VMs, containers, and unikernels. Multiple performance parameters were analyzed such as network bandwidth, memory footprints, CPU utilization, etc., and the authors found that the performance of containers was much better but there were also some issues such as security, data management, and networking overhead in a large cluster deployment, nonstable performance, and a smaller number of tools for container management and orchestration.

Group buying strategies proposed by Yi et al. [19] were used to address cloud inefficiency and improve resource utilization. The proposed mechanism was named Cocao (COmputing in COntAiners) which was used to group the jobs according to resource demands and was assigned to the predefined cloud provider's group buying deals. Each deal offered the pool of resources for the job belonging to that group. The authors modeled the initial group organization as a variable-sized vector bin packing problem and used an online multidimensional knapsack for dynamic group organization to fill the resource holes.

Khan et al. [20] addressed the issue of consolidating virtual machines, containers, and/or applications to have an energy- and performance-effective schema for heterogeneous cloud data centers. According to the authors, there is a trade-off involved between migrating containers and virtual machines. The experimental results demonstrated that container migration increases the total number of migrations when compared with VM migrations (due to the small size of containers). The VM migration is more performance-efficient, while container migration is energy-efficient. The proposed model evaluation showed that a large number of container migrations result in high energy utilization and a decline in performance when compared to VM migration.

The study in [21] presented two types of ECSched scheduling strategies, i.e., ECSched-dp (based on the dot-product heuristic) and ECSched-ml (based on most loaded heuristic). Moreover, the authors compared ECSched with Google Kubernetes and Docker Swarm which acted as the baseline for comparison. The comparison results showed that ECSched provides more resource utilization and ECScheddp has the highest resource utilization. The resource utilization increased by 4.1% to 5.3% for ECSched-dp when compared with baselines.

To reduce the service cost, Chung et al. [22] introduced Stratus, which was used to pack the tasks on the machines using the estimated runtime for the incoming jobs. The JVuPredict [23] algorithm was used to predict the estimated runtime of the tasks. JVuPredict determined the candidate group of the incoming job based on the history (i.e., submitted by the same user, same job name, time of day, etc.), and then the tentative runtime was evaluated. The Stratus model made the resource allocation of the machines which were either highly utilized (mostly full) or empty ones (i.e., can be released to save cost). The simulation results showed that cost can be decreased by 17 to 44%. However, in the experiment, it was assumed that there are no intertask dependencies and that the task

co-location has no or minimal impact on task runtime, which is not true for the real-world cloud computing environment.

A framework named MiCADO was proposed by Visti et al. [24], which provided application-level dynamic cloud orchestration, thus resulting in predicting the application capacity demand behavior. The microservices orchestration layer was divided into four sublayers; refer to Figure 3. The cloud interface layer provided cloud access from the above layers. The microservices coordination layer monitored the current execution performance and helped in identifying the bottleneck or under-utilization of infrastructure. The microservices discovery and execution layer managed the microservice execution and kept track of services running, i.e., service name, IP address, and port number. The coordination interface API provided access to application developers for the convenient development of the dynamically scalable cloud-based application. The framework was presented as the first proof-of-concept and its applicability was checked using open-source tools such as Docker1, Consul2, etc.

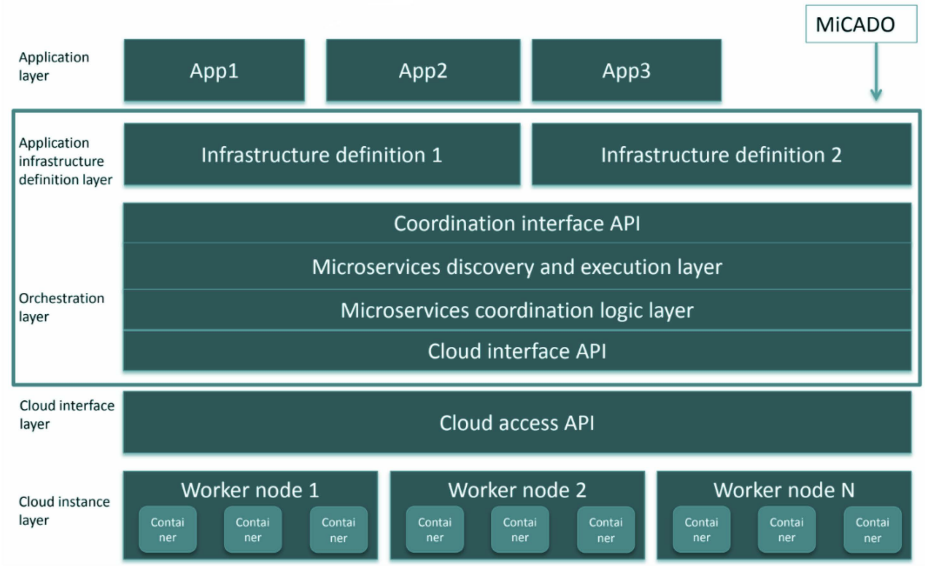

Figure 3. MiCADO Architecture; Adapted with permission from [24].

In 2019, the MiCADO by Visti et al. [24] was implemented by Kiss et al. [25], which provided support for the automated scalability of cloud applications. The implementation for evaluation of MiCADO was based on the CloudSigma public cloud. Moreover, open-source tools, such as Prometheus (to collect information about various services), Occopus (to keep track of running services), Consul (to organize Docker containers in the cluster), and Grafana (to generate graphs), were used. They estimated that the proposed framework can increase the efficiency of the business process and help in increasing customer satisfaction.

To support the deadline-based application execution policies for MiCADO, a cloud-agnostic queuing system was introduced by Kiss et al. [26]. This study was based on MiCADO as already described by Visti et al. [24], which was implemented by Kiss et al. [26]. The study implemented the JQueuer system, which made use of the queuing system to schedule many jobs among the containers. The JQueuer was based on two components named: JQueuer Manager and JQueuer AGent. JQueuer agent components were responsible for fetching the jobs from the job queues and sending them to JQueuer Manager. The study also provided the implementation of JQueuer along with MiCADO to provide deadline-based execution policies for the MiCADO framework. The paper mainly focused on the demonstration of JQueuer and MiCADO and did not address the optimization of deadline-based execution policies.

Another research study [27] defined a general framework to provide a programmable policy keeper concept that performs the decision about scaling at the VM level and con-

tainer level by calculating the optimal number of required instances. The framework was integrated with MiCADO and tested with the deadline-based scaling use case. However, the future plan is to provide the machine learning algorithm to support the policy keeper component.

A migration service called Voyager was introduced by Nadgowda et al. [28]. This service provided just-in-time container migration to minimize downtime. The core idea was to resume the operations instantly on target machines while doing the disk state transfer in the background. However, the implementation heavily relied on the open-source Linux tool CRIU for checkpoint/restore operations. In the Voyager migration, the downtime was realized between 2 and 3 s, which is relatively high. In addition, the resources optimization and prioritization were not handled for better results.

A study by [29] presented a three-phase virtual resource management framework for energy-efficient resource management of cloud data centers. In the first phase, profiling is performed to generate machine-readable data sets of computing tasks. The task profiling gave information about the required number of PMs. In the second phase, the tasks are classified into long-running, normal, and tiny tasks. The classification of task running length is used to assign the tasks to respective VMs. In phase three, all PMs are sorted based on energy efficiency. Then the PMs which are more energy-efficient are first loaded with VMs. The simulation results showed that 8–12% energy savings could be achieved by using the proposed framework. For this framework to be successful, we need to know the size of the task beforehand. Moreover, it does not address the holes in PMs, which can be formed by assigning a few PMs only for large tasks and some PMs only for small tasks.

Zhou et al. [30] proposed the optimal placement schema for containers in the cloud environment. They introduced a one-shot algorithm that works as a placement schema for the container cluster and an online algorithm that breaks down the online decision making into on-the-spot decisions depending on resource price.

The study by Tan et al. [31] provided an extension of the NSGA-II-based approach [32] for service resource allocation in the cloud. The study aimed at the static approach instead of the dynamic approach for allocation of the container to the virtual machines and physical machines. Instead of using a single chromosome [32], the concept of a dual-chromosome genetic algorithm was introduced for resource allocation in container-based clouds. The results showed that dual-chromosome GA gives better performance when compared with single-chromosome GA and the best-fit descending algorithm.

As most of the elasticity policies are based upon the threshold-based heuristics algorithm, the study by [33] explored and designed a more flexible solution using reinforcement learning (RL) to control the elasticity of container-based applications.

Technically, the conventional service-oriented architecture (SOA) services and microservices are platform-agnostic and use standardized communication protocols such as HTTP. The microservice architecture divides the monolith (single-unit) software into dozens of small service components. The microservice is highly scalable because each service can be scaled easily when required. The individual nature of microservices means they are easily usable among the different systems.

The number of microservices may grow to hundreds or even thousands in number. The death star graphic visualization [34] for a social media platform is shown in Figure 4. The complexity of many microservice interactions is evident. The increasing number of microservices brings complexity to the management of services. As the network of microservices grows, it also brings challenges for interservice communication.

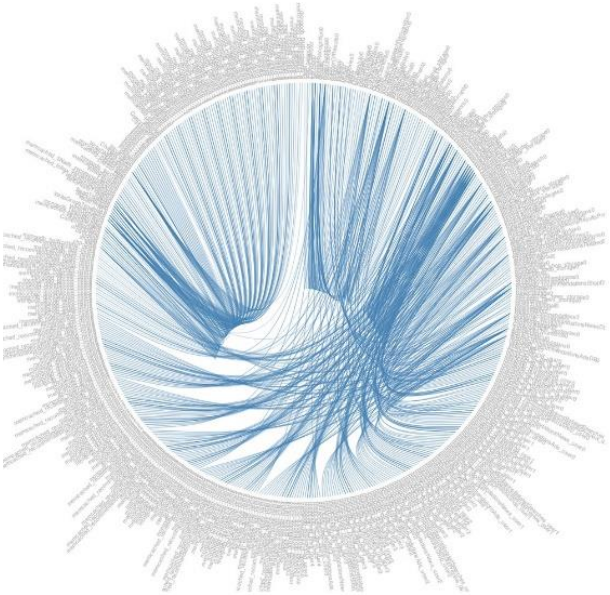

Figure 4. A death star graph showing inter-relationships among microservices [34].

A microservices tracing and analytical tool (JCallGraph) to analyze the invocation relationship of microservices running on containers was designed by Liu [35] for the Chinese e-commerce company JD.com (having approximately 34,000 microservices running on a cluster of 500,000 containers, supporting over 250 billion RPC-based microservices calls per day). The JCallGraph acted as a distributed tracing system to track the interactions and timing information of microservices.

Dang-Quang and Yoo [36] suggested an effective multivariate autoscaling system for cloud computing utilizing bidirectional long short-term memory (Bi-LSTM). The monitor–analyze–plan–execute loop served as the foundation for the system structure. In terms of prediction accuracy, the assessment results reveal that the proposed multivariate Bi-LSTM model outperforms not only the univariate Bi-LSTM model but also multivariate deep learning models such as LSTM and CNN-LSTM.

Malik et al. [37] addressed resource consumption prediction to avoid over-provisioning and under-provisioning problems. The purpose of this study was to anticipate multire-source consumption utilizing a functional-link neural network (FLNN) with a hybrid genetic algorithm (GA) and particle swarm optimization (PSO). The model trains network weights using a hybrid GA-PSO algorithm and for prediction uses FLNN. When compared to conventional methodologies, the hybrid model offered higher accuracy.

Malik et al. [38] addressed the issue of energy consumption and resource usage efficiency in virtualized cloud data centers. The workflow activities were preprocessed in the suggested approach to prevent bottlenecks by putting jobs with higher dependencies and lengthier execution durations in different queues. The tasks are then categorized based on the intensity of the needed resources. Finally, the optimum schedules are chosen using particle swarm optimization (PSO). The results of the tests demonstrated efficacy in terms of energy usage, makespan, and load balancing.

Calderón-Gómez et al. [39] evaluated the service-oriented architecture patterns and microservice architecture patterns for the deployment of applications in cloud computing, specifically the eHealth applications. It was determined that the microservice architecture performs better in terms of response time and performance than the service-oriented architecture variant; nevertheless, the microservice architecture consumes much more bandwidth than the service-oriented architecture variant.

Górski and Woźniak [40] conducted a thorough evaluation of the literature on the optimization of business process execution in services architecture. They classified the methodologies in the existing literature into three categories, resource allocation, service

composition, and service scheduling, and discovered that service composition draws the most researchers. In general, scientists offer heuristic strategies for optimizing business operations in real time and still there is room for investigation at the resource allocation and service scheduling levels, which are covered in this article.

Due to the high demand for cloud computing services, the cloud service providers must work together in a scenario known as an interconnected cloud computing environment [41]. Commonly known interconnected cloud environments are hybrid cloud, intercloud, federated cloud, and multicloud. The cloud service provider and the cloud service user agree upon a service level agreement (SLA). If the services are not provided according to the agreement, then the cloud provider has to pay certain plenty. The studies have been conducted to increase the overall efficiency of cloud services to decrease SLA violations. Moreover, different approaches have been defined to customize the SLA according to the present status of data centers.

Hemmat and Hafid [42] addressed the topic of SLA violation prediction in the cloud computing environment and used the machine learning approach for SLA violation prediction. The study compared the performance of the naive Bayes model and the random forest model. They found that the random forest model provided the best performance with an accuracy of 0.99%. The classical way of predefined SLA has certain issues. The predefined SLAs are not able to change according to the change in QoS parameters. SLA can be defined as a set of Service Level Objectives (SLO). These SLOs should be ensured by the cloud service provider according to the changing status of the data centers.

The key findings of the literature are highlighted in Figure 5. The literature showed that since cloud computing maintains a range of virtualized resources, scheduling is an essential component. Inefficient scheduling strategies and nonoptimal resource management confront the issues of resource overutilization and underutilization, thus resulting in resource imbalance, which results in either impairment in cloud service performance, i.e., overutilization, or waste of cloud resources, i.e., underutilization. In addition, efficient SLA and computational monitoring infrastructures are still lacking which must be capable of monitoring and detecting SLA violations. Monitoring, detection, and pre-emption of SLA should be imposed to provide better resource availability, scalability, competitive price, and energy efficiency for cloud services.

Notably, scientific research highlights two sorts of rating strategies, namely, profiling and ranking, for effective content distribution of interconnected services using the repairing genetic algorithm (RGA) [43], bubble-up and bubble-flux [44], the power-conscious model [45], service request routing optimization [46], resource balance ranking schema [47], ge-kube using Kubernetes for geographical distributions [48], energy usage in relation to processing intensive communication workloads [49], StressCloud to monitor cloud energy consumption [50], the positioning of microservice instances and resource distribution optimization [46], and CloudCost which used a UML profile to allow the modeling of user behavior and cloud architecture [51].

According to the literature investigation, most cloud services have certain limits, such as a maximum cost, task complexity, maximum completion time, less profit, and make span. Based on a review of the current literature, it is also discovered that numerous container orchestration solutions and provisioning techniques do not consider the unique properties of microservices such as high cohesion, independence, autonomy, and loose coupling. Open key research areas can be listed as:

- Distribution techniques in container orchestration platforms are mostly focused on available and allocated resources and do not take into account the complexity of the microservices' architecture, microservices' properties, and workload management accordingly, which need to be addressed.
- The ability to effectively forecast compute demand and application performance under varying resource allocations is one of the research issues related to elastic services.
- Dynamic container orchestration according to microservices interactivity patterns and concerning entities is an active research topic that has room for additional investigation.

- Containers have less isolation and security than virtual machines (VMs) due to kernel sharing, which is an open issue but not addressed in the article due to the study's focus and limitation.

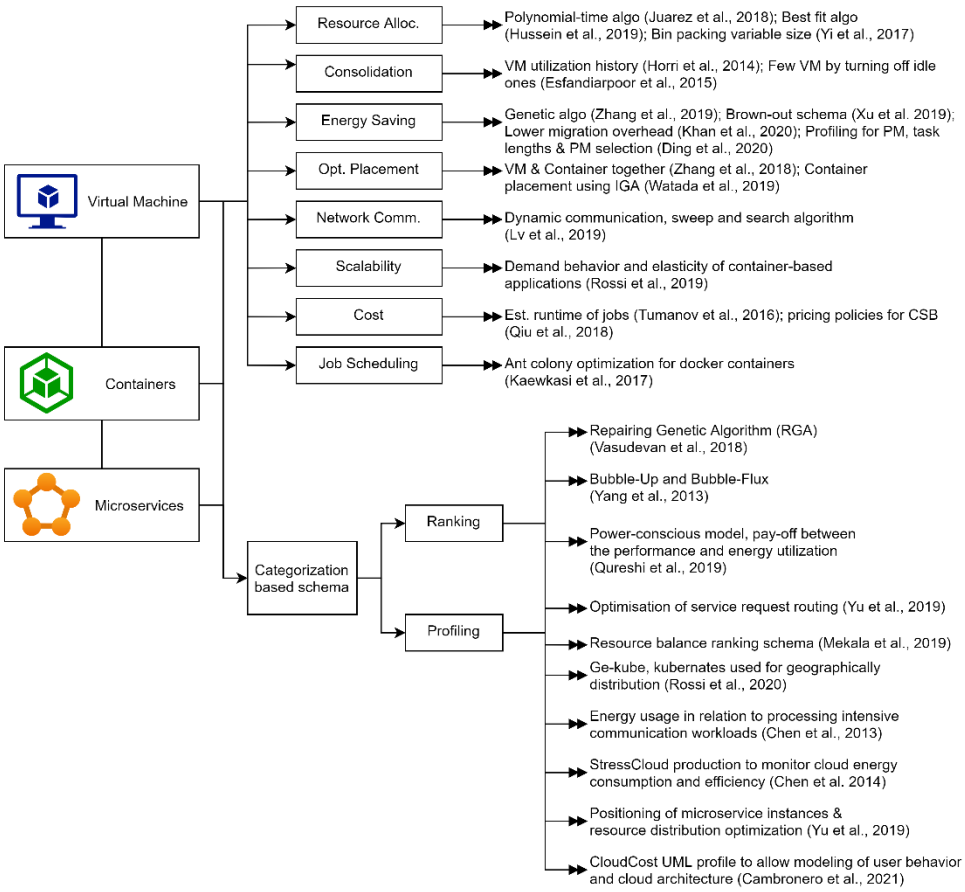

Figure 5. Key findings from the literature review of concepts, theories, and existing frameworks [6–15,18–20,23,29,33,43–51].

To overcome the aforementioned limits while maintaining the microservices' characteristics, a new generation of the framework must be developed. As a result, futuristic research on task scheduling and resource management in cloud computing must focus on developing better scheduling algorithms and frameworks based on multiobjective functions and adding additional factors such as mutation action policies to increase the performance and lower the energy consumption of the cloud-based applications. Therefore, this study specifically contributes to the following:

1. Intelligent partitioning: responsible for microservice classification and the categorization of microservices based on their usage and dependency on other microservices.
2. Dynamic allocation: used for the pre-execution distribution of microservices among containers and making decisions for the dynamic allocation of microservices at runtime.
3. Resource optimization: in charge of shifting workloads and ensuring optimal resource use.
4. Mutation actions: these are based on procedures that will mutate the microservices based on cloud data center workloads.

The next section (Section 3) discusses the mechanics and details of the above-mentioned contribution highlights.

3. Methodology

The research methodology is based on a number of necessary steps including literature review, feature selection, data collection, agent definition, and evaluation; refer to Figure 6.

The research process performed an orderly and critical review of existing studies. The study of past research papers about the latest trends of cloud computing acted as a data collection tool to gain knowledge about the existing frameworks and results achieved. Based on the literature review understanding, the system features were identified and later classified into different sets of classes. The core feature selection greatly helped in meaningful decision-making. The newly conceptualized techniques were formed/extended to address the untouched issues of cloud computing. The eco-friendly dynamic resource discovery and resource allocation models were identified to meet the computational demands of the cloud jobs.

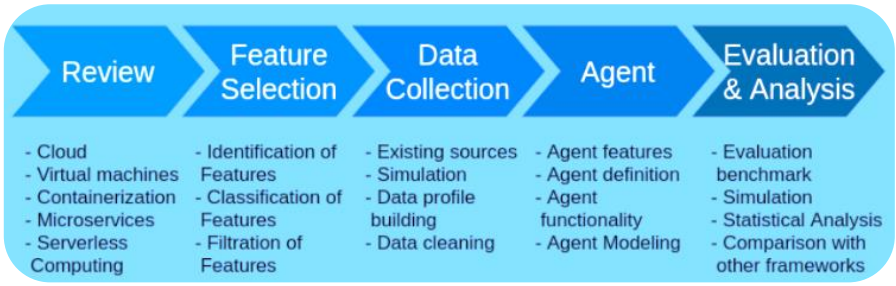

Figure 6. Research methodology flow.

It became evident from the selected feature that existing techniques to achieve better resource allocation and reduce the power consumption of the computing nodes in a cloud environment have some serious deficiencies. Most of these techniques monitor the resource load over a period of time and decide to turn off the computing node to reduce the power consumption. Moreover, the main scheduler and resource handler run on the master node, which defines the critical bottleneck entry and a single point of failure (SPF) in the computing environment. Mostly the defined systems only consider the CPU allocation while ignoring the other parameters. The latency due to changes in the state of machines (on/off) is also not considered.

Our research model is based on the microservice architecture in which a cloud application is divided into several small services that will be capable enough to perform the signal functionality. A prototype of a microservices-based system is shown in Figure 7, where multiple actors access the database server and web server through API gateway, frontend interfaces, and storage gateways. In between, multiple independently running microservices provide the required service while being able to handle millions of access requests and ensuring low latency, high availability, scalability, and resilience to network failures. To handle a large number of microservices in the system, we proposed an agent-based service (see Figure 8) which will be responsible for providing four core services including intelligent partitioning, dynamic allocation, resource optimization, and mutation actions.

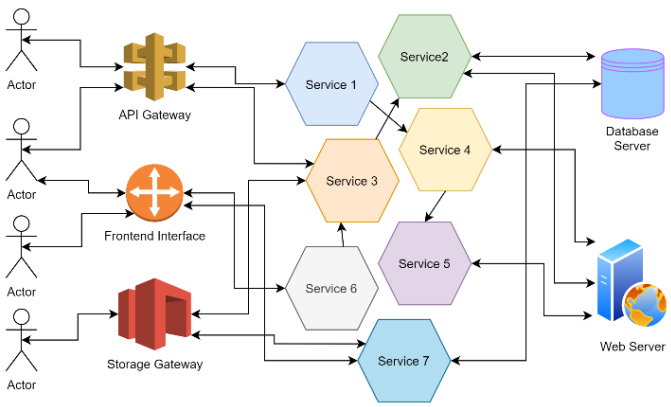

Figure 7. A microservice system prototype.

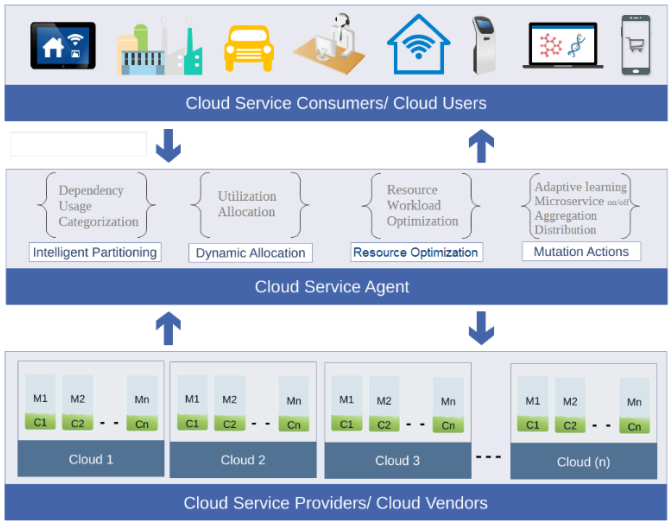

Figure 8. Layered representation of the proposed framework.

3.1. Intelligent Partitioning

The intelligent partitioning module is responsible for the categorization of microservices based on their usage and dependency on other microservices, while making use of microservices' autonomy, loose coupling, and scalable nature to locate the microservices closer to the data source to reduce latency and network utilization.

Each job in a microservice is allocated a thread as a resource. Local databases in microservices offer cross-thread connections, which implies that many threads in microservices can utilize the same database connection [52]. In such a case, the computing resources available will limit the number of CPUs and other resources required for the operation. The intelligent partitioning agent has autonomous decision-making capacity based on the reliance and consumption of the microservices to assess and group the microservices based on the present condition of the cloud application. The aggregated microservices are then assigned to available data centers. As a result, it minimizes latency and network use, improves resource utilization, and addresses service discovery and load balancing difficulties.

The overall schema for intelligent partitioning is represented in Figure 9. The total number of microservices of a specific application will be grouped on one of two factors, i.e., (1) based upon connections/dependencies among microservices, (2) the usage of the history of the microservices invoked by the actors involved (actors can be end-users, web services, REST API, RPC call, etc.). The grouping functionality is provided by the running agent. The agent may make use of developer documents or the service usage history logs of invoked microservices.

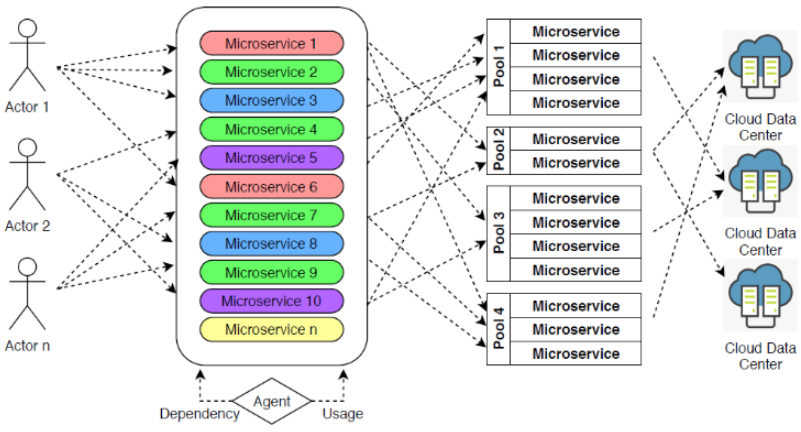

Figure 9. Schema for intelligent partitioning.

3.2. Dynamic Allocation

This module is responsible for the allocation of containers and the respective microservices. At run time, the decisions about allocations are made according to the changing behavior of the cloud services utilization. Within containerized cloud environments, the service instances have been assigned to network locations dynamically. The autoscaling and upgrades result in continuous change in service instances' network locations. Thus, a service registry mechanism is used to enable the service to discover [53]. The service registry stores the network locations (IP address and port numbers) of service instances. The service registry, therefore, records the network location when a service instance starts up and removes the network address entry when the service instance terminates. The dynamic allocation makes use of the service registry to obtain knowledge about existing allocations and to upgrade the service registry entries according to the newly developed allocation table.

Moreover, in software development, the updates for programming codes are automatically built, tested, and deployed, thus resulting in the continuous delivery and deployments of the software updates. Thus, it can use the Git logs for changing the allocations according to software updates. In some cases, the dynamic approach tends to produce worse results when compared to the static approach [31,54]. This is due to frequent container shifts by the dynamic approaches, which leads to extra network overhead. Thus, the dynamic approaches in our proposed framework shift the containers intelligently to reduce the network traffic among the containers.

3.3. Resource Optimization

Fulfilling the demands of containers for resources can be challenging. The cluster of containers has multiple resource needs (such as CPU and memory) which need to be allocated efficiently. In addition, poor/good container placement on the physical machines (PM) can affect the network latency of container communication. In large container clusters, performance parameters such as scaling out and recovery from failure require real-time analytics [30,55]. The queue-based tasks and resource scheduler are commonly used in well-known cloud orchestration platforms such as Docker Swarm and Kubernetes. In such schedulers, the task placement request on the cloud enters a queue. From the queue, the scheduler fetches the request and processes one container at a time. Most commonly, the variants of heuristic packing algorithms, such as first-fit decreasing (FFD) and best-fit decreasing (BFD), are used for queue-based schedulers.

In the proposed framework for resource optimization, the migration service is introduced, which will be responsible for shifting the workload between containers and providing optimal resource utilization of computational resources; refer to Figure 10. There could be many reasons to migrate the containers from one host to another host. A few of these possibilities include:

- The overall work needs to be distributed across multiple hosts to comply with the service level agreement.
- Consolidating the number of hosts; thus, a smaller number of hosts will result in lower consumption of energy.
- A specific host performs better than the currently running host.

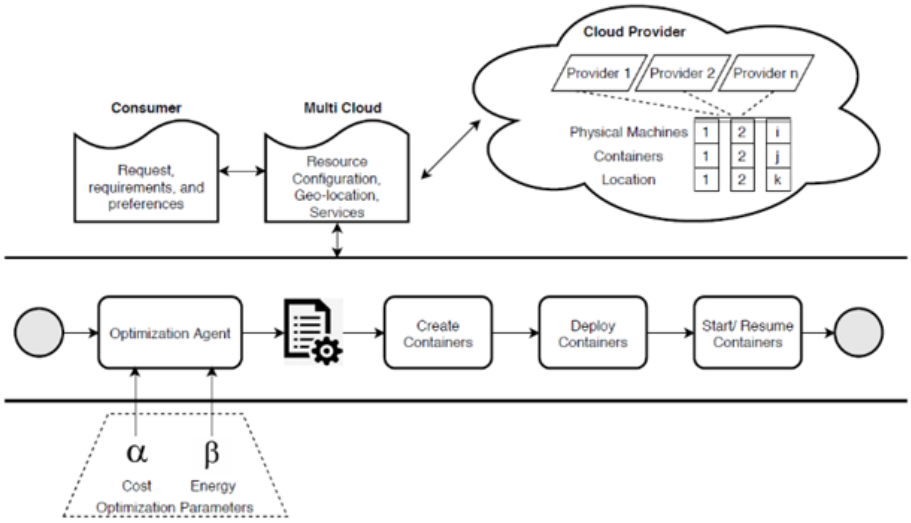

Figure 10. Resource optimization schema.

3.4. Mutation Actions

These actions are based upon mutation processes. The mutation is defined as: "A mutation is a change in the DNA sequence. Mutations can develop because of multiple environmental influences". Thus, the mutation operator involves transferring procedures from one microservice contender towards another [56]. In our proposed model, the change in container orchestration occurs according to the defined system QOS parameters. Either the microservices will be turned off or on, replicated due to workload, aggregated, or distributed according to geographical carbon footprints, thus giving an impression of mutation.

To identify the mutable microservices, an identifier system will work in coordination with other modules, which will highlight the candidate microservices. The mutation action schema is shown in Figure 11. The microservices running on containers will generate a microservices invocation log. The data mining module will find the patterns and correlations within the log data sets to predict/identify the candidate microservices for mutation actions. The outcome of the data mining module will be used by the event handler. When an event is generated (due to factors such as overloaded resources), it will be received by the event handler. The event handler will mutate (in this case, shut down the microservices, shown in black color in Figure 11) the microservices.

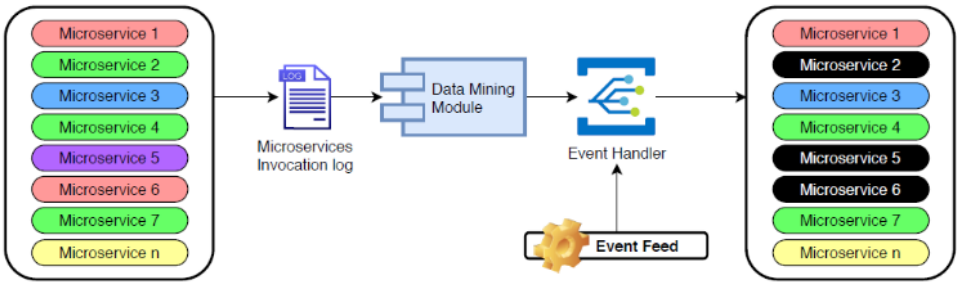

Figure 11. Mutation actions schema.

In the classical cloud computing approach, the task is assigned within one data center. However, in our approach, the task distribution strategies are applied in interconnected cloud computing environments including multicloud, hybrid cloud, and federated cloud environments. Thus, an agent-based cloud broker service layer is responsible for managing the incoming request from cloud users and providing the delivery of cloud services. It will be responsible for the performance and will also act as a negotiator between the cloud

service providers and cloud service consumers. The negotiation terms involve SLA, cost, energy, and other details. In a more effective approach, the scheduling strategies are applied across multiple data centers of one vendor and across multiple data centers of different vendors.

Moreover, event-based architecture is introduced, in which a service sends an event to the event observer and one or more event observer containers that were watching for that event to respond to the specific event. The response time for cold versus warm service requests has a major impact on the performance of cloud providers. The response time implication is evaluated for three situations, i.e., (1) a provider cold request, the very first call to the microservice made to the cloud provider; (2) a container cold request, the very first call made to the container hosting the microservice; and (3) a warm request, the repeated request to the already invoked container hosting the microservice.

3.5. Metrics for Framework Performance Assessment

Maintaining business operations and long-term sustainability significantly depends on cloud performance. Cloud performance metrics enable effective monitoring of cloud resources, operational efficiency, and service delivery. There are a variety of metrics that can assist in monitoring and assessing the performance of cloud computing services. Typically, cloud performance indicators include VM capacity (refer to Equation (1)), multiple VM instances' capacities (refer Equation (2)), task execution time (refer to Equation (4)), energy consumption (refer Equation (5)), resource utilization (refer to Equation (6)), and power consumption (refer Equation (7)). Details of some of the notations from the literature are listed below:

Capacity of a Virtual Machine (VM): A virtual machine's capacity is the product of the number of cores allotted to every virtual machine as well as the size of each core. Let V be the set of VM instances represented as $V_1, V_2, V_3, \ldots, V_n$. Then the capacity of a virtual machine CP_V is defined as:

$$\mathrm{CP_V} = \mathrm{C_{num}(VM_V)} \ * \ \mathrm{C_{units}} \tag{1}$$

Capacity of Multiple VM Instances: The VM instances' capacity is defined as the total of the active VM instances on the server S:

$$\mathrm{CP_{sum}} = \sum_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{CP_i} \tag{2}$$

Note: The overall volume of active virtual machine instances should also not exceed the server's (S_p) capacities and therefore can be represented in Equation (3):

$$\mathrm{CP_{sum}} \leq \mathrm{S_p}; 1 \leq \mathrm{p} \leq \mathrm{n} \tag{3}$$

Execution Time: The task (Tk) execution time (ET) by the virtual machine instance VM_i is defined as:

$$\mathrm{ET_{ki}} = \frac{\mathrm{SZ_k}}{\mathrm{CP_i}} \tag{4}$$

where the size of the task k is denoted as SZ_k.

Energy Consumption: Energy consumption can be defined as:

$$\mathrm{EC} = \mathrm{EC_p} + \mathrm{EC_t} + \mathrm{EC_m} + \mathrm{EC_e} \tag{5}$$

where CPU energy consumption is EC_p, switching equipment energy consumption is EC_t, storage devices' energy consumption is EC_m, other units' energy consumption, including current conversion losses, are represented as EC_e.

Resource Utilization: It is represented as the ratio of the execution time of a workload executed by a particular resource and the total uptime of the resource:

$$RU = \sum_{i=1}^{n} \frac{\text{execution time of a workload executed on resource i}}{\text{total uptime of resource i}} \quad (6)$$

Power Consumption: Power utilization can be defined as:

$$P(u) = k \cdot P_{max} + (1 - k) \cdot P_{max} \cdot u \quad (7)$$

where P_{max} is the maximum power consumed when the server is fully utilized; k is the fraction of power consumed by the idle server, and u is the CPU utilization.

4. Results and Discussion

The goal of the study is to conceive a conceptual framework that assists in the containerized microservices orchestration and provisioning in cloud computing environments. To associate the concepts of resilience and applicability, a part of the framework in a custom-built simulation environment has been simulated and the preliminary results of the limited framework evaluation have been presented.

As previously stated, the proposed framework is comprised of four levels. To narrow the test, the evaluation of an intelligent partitioning module that uses the dependency agent has been made. The microservice architectural style organizes an integrated suite of tiny independent services that are designed for a business domain. To simulate the intelligent portioning, the microservices were grouped according to their design patterns [57]. The major categories of the design patterns for microservices are decomposition patterns, observability patterns, database patterns, integration patterns, and cross-cutting concern patterns. For implementation, the integration of domain-driven design (DDD) subdomain aggregates was intended. A domain (e.g., e-commerce business) is made up of many subdomains (inventory, orders, delivery, customers, etc.). Each subdomain represents a distinct aspect of the business. The aggregated items of the subdomain can be viewed as a single entity. This results in eliminating the possibility of object references exceeding service borders and satisfies the limits of the microservices transaction model since a transaction may only add or edit a single aggregate. In such a configuration, clustering and assigning individual containers to microservices is used when a service relies on the delivery of another service. These assemblies of containers are subsequently assigned to the appropriate cloud data centers. The performance efficiency of design-pattern-based intelligent portioning and distribution of microservices has been compared with the arbitrary distribution of microservices where the containers along with microservices were randomly assigned to available cloud data centers. As a result, for the distribution of microservices on cloud data centers, pattern-based clustering of microservices was not conducted in the arbitrary distribution design.

A custom-built simulation program created microservice data sets of microservice consumption to verify the efficiency of intelligent partitioning. The simulation setup generated a data collection of microservices calls spread across 750 min, utilizing 150 microservices distributed across three data centers in three distinct geographical regions. Figure 12 depicts the average response time of a collection of microservices that are arbitrarily distributed. The average response time for microservice execution was 623.83 milliseconds; the minimum and highest response times were 237.28 and 1228.16 milliseconds, respectively.

The intelligent partitioning employing a design-pattern-based distribution approach has been analyzed using the same simulation environment. The findings, as shown in Figure 13, indicate that response time is significantly reduced when compared to the arbitrary distribution method. The average response time for microservice requests is 457.45 milliseconds. The minimum response times for simulated service calls are 197.55 ms and the maximum value of response time is 1228.71 ms.

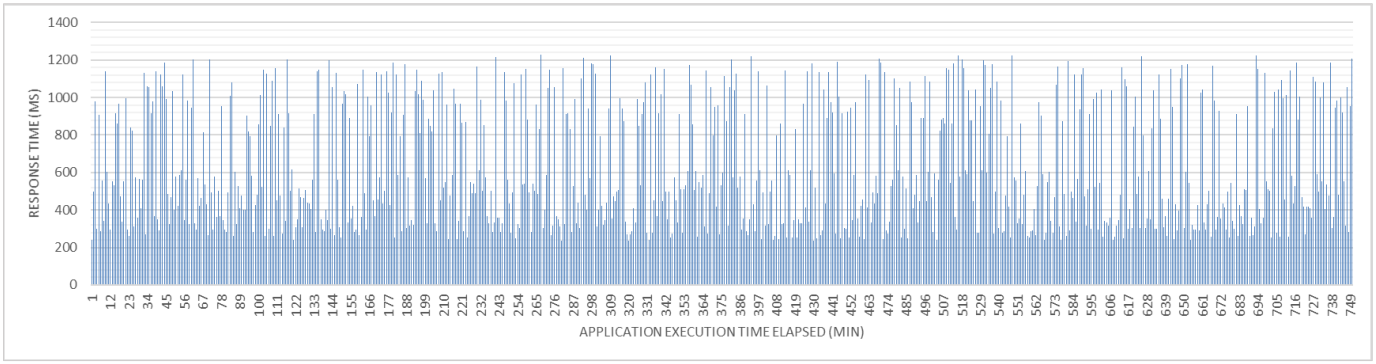

Figure 12. Response time over a period of 12.5 h when microservices are deployed using the arbitrary distribution model.

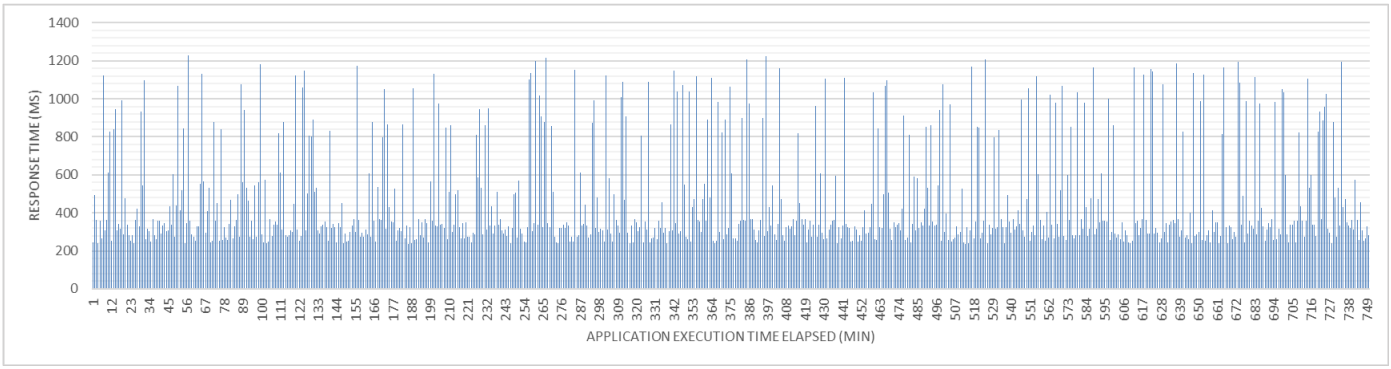

Figure 13. Response time over a period of 12.5 h when microservices are deployed using the design-pattern distribution.

The resource optimization part of the proposed framework was tested and evaluated in a separate study using custom-built simulation which is used for modeling and simulation of microservices in cloud computing environments. The details of which are out of scope for this study, as this study mainly defined the conceptual framework for cloud orchestration and provisioning based on container-level microservices.

Briefly, we can state that dynamic provisioning of containers and respective microservices were tested and evaluated using a computing cloud where the user-defined number of data centers was established for a real evaluation of the part of the framework schema. The simulation findings showed that the predistribution of microservices depending on their levels of engagement activities greatly decreases carbon dioxide emissions while increasing green energy use [58]. The usage of cutting-edge container microservice architecture required the use of several containers scattered across various data centers in the cloud environment. The suggested microservice ranking technique [58] allowed services to be separated from one another, resulting in groups of microservices that may be hosted in geographically scattered data centers. It infers that a rank-based distribution lowers the communication latency considerably.

It however, requirs performing intensive modeling and simulation of containerized computing environments using a software package named ContainerCloudSim [59], which is an extension of CloudSim. Moreover, to test in the real-world environment, it needs to use DeathStarBench [60] which is an open-source benchmark suite for cloud microservices. DeathStarBench provides end-to-end services including social network, hotel reservation service, and media service. The proposed conceptual framework will be tested on well-known input parameters and variables and deployment scenarios (hybrid cloud/multicloud/federated cloud). The key evaluation parameters will include power

usage efficiency (PUE), server utilization rate, server refresh rate, virtualization ratio, and carbon footprint.

5. Conclusions

This study presents a conceptual framework to manage the high number of microservice execution while reducing the response time, energy consumption, and execution costs. The framework's four core services have been described as:

- Intelligent partitioning, which is in charge of microservice classification.
- Dynamic allocation, which is used to distribute microservices to containers at the start of the service and then make dynamic microservice allocation decisions at runtime.
- Resource optimization, which will be in charge of redistributing workloads and maximizing resource use.
- Mutation action, which is built on procedures that will change microservices in response to cloud data center workloads.

We used a custom-built simulation environment for the evaluation of an intelligent partitioning module that leveraged design patterns, i.e., when a microservice relies on other microservices for service delivery, they are clustered together and allocated to separate containers. The performance of the integration of domain-driven design (DDD) based intelligent portioning indicates that response time is significantly reduced when compared to the arbitrary distribution method.

In future work, the proposed framework will be evaluated and tested (as described in the previous section) using a simulation environment that includes CloudSim, for the modeling and simulation of cloud computing, and ContainerCloudSim, for the modeling and simulation of containerized cloud computing. Furthermore, expert scheduling techniques and learning systems for containers and microservices would be integrated and tested on a real cloud container cluster to lower the temporal complexity.

Author Contributions: Conceptualization, A.S. and M.F.H.; methodology, A.S.; software, A.S.; validation, M.F.H., R.A., S.A.M. and S.N.M.S.; formal analysis, A.S., S.A.M. and F.H.; investigation, A.S., M.F.H., R.A. and M.A.S.; resources, M.F.H.; data curation, A.S.; writing—original draft preparation, A.S.; writing—review and editing, A.S., R.A., S.N.M.S., M.A.S., S.A.M. and F.H.; visualization, A.S., F.H., M.A.S. and S.N.M.S.; supervision, M.F.H.; funding acquisition, M.F.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Fundamental Research Grant Scheme (FRGS), reference number: FRGS/1/2020/SS02/UTP/03/1, under the Malaysia Ministry of Higher Education, grant cost centre number "015MA0-057", and the Center for Graduate Studies (CGS), Universiti Teknologi PETRONAS (UTP), Perak, Malaysia.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: We acknowledge the support, resources, and computing environment provided by the High-Performance Cloud Computing Center (HPC3), Universiti Teknologi PETRONAS (UTP), 32610 Seri Iskandar, Perak Darul Ridzuan, Malaysia.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mell, P.M.; Grance, T. *The NIST Definition of Cloud Computing*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2011. [CrossRef]
2. Buyya, R.; Yeo, C.S.; Venugopal, S. Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. In Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications, Dalian, China, 25–27 September 2008; pp. 5–13.
3. Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A.D.; Katz, R.; Konwinski, A.; Lee, G.; Patterson, D.; Rabkin, A.; Stoica, I.; et al. A View of Cloud Computing. *Commun. ACM* **2010**, *53*, 50–58. [CrossRef]

4. SaaS vs. PaaS vs. IaaS: What's The Difference & How To Choose—BMC Software | Blogs. Available online: https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/ (accessed on 11 February 2022).
5. Gai, K.; Qiu, M.; Zhao, H.; Tao, L.; Zong, Z. Dynamic Energy-Aware Cloudlet-Based Mobile Cloud Computing Model for Green Computing. *J. Netw. Comput. Appl.* **2016**, *59*, 46–54. [CrossRef]
6. Xu, M.; Buyya, R. BrownoutCon: A Software System Based on Brownout and Containers for Energy-Efficient Cloud Computing. *J. Syst. Softw.* **2019**, *155*, 91–103. [CrossRef]
7. Juarez, F.; Ejarque, J.; Badia, R.M. Dynamic Energy-Aware Scheduling for Parallel Task-Based Application in Cloud Computing. *Future Gener. Comput. Syst.* **2018**, *78*, 257–271. [CrossRef]
8. Qiu, C.; Shen, H.; Chen, L. Towards Green Cloud Computing: Demand Allocation and Pricing Policies for Cloud Service Brokerage. *IEEE Trans. Big Data* **2018**, *5*, 238–251. [CrossRef]
9. Horri, A.; Mozafari, M.S.; Dastghaibyfard, G. Novel Resource Allocation Algorithms to Performance and Energy Efficiency in Cloud Computing. *J. Supercomput.* **2014**, *69*, 1445–1461. [CrossRef]
10. Esfandiarpoor, S.; Pahlavan, A.; Goudarzi, M. Structure-Aware Online Virtual Machine Consolidation for Datacenter Energy Improvement in Cloud Computing. *Comput. Electr. Eng.* **2015**, *42*, 74–89. [CrossRef]
11. Watada, J.; Roy, A.; Kadikar, R.; Pham, H.; Xu, B. Emerging Trends, Techniques and Open Issues of Containerization: A Review. *IEEE Access* **2019**, *7*, 152443–152472. [CrossRef]
12. Zhang, R.; Chen, Y.; Dong, B.; Tian, F.; Zheng, Q. A Genetic Algorithm-Based Energy-Efficient Container Placement Strategy in CaaS. *IEEE Access* **2019**, *7*, 121360–121373. [CrossRef]
13. Zhang, R.; Zhong, A.; Dong, B.; Tian, F.; Li, R. Container-VM-PM Architecture: A Novel Architecture for Docker Container Placement. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer: Cham, Switzerland, 2018; Volume 10967, pp. 128–140, ISBN 9783319942940.
14. Lv, L.; Zhang, Y.; Li, Y.; Xu, K.; Wang, D.; Wang, W.; Li, M.; Cao, X.; Liang, Q. Communication-Aware Container Placement and Reassignment in Large-Scale Internet Data Centers. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 540–555. [CrossRef]
15. Hussein, M.K.; Mousa, M.H.; Alqarni, M.A. A Placement Architecture for a Container as a Service (CaaS) in a Cloud Environment. *J. Cloud Comput.* **2019**, *8*, 7. [CrossRef]
16. Srikantaiah, S.; Kansal, A.; Zhao, F. Energy Aware Consolidation for Cloud Computing. In Proceedings of the Conference on Power Aware Computing and Systems, Berkeley, CA, USA, 7 December 2008.
17. Tchana, A.; Son Tran, G.; Broto, L.; DePalma, N.; Hagimont, D. Two Levels Autonomic Resource Management in Virtualized IaaS. *Future Gener. Comput. Syst.* **2013**, *29*, 1319–1332. [CrossRef]
18. Kaewkasi, C.; Chuenmuneewong, K. Improvement of Container Scheduling for Docker Using Ant Colony Optimization. In *Proceedings of the 2017 9th International Conference on Knowledge and Smart Technology: Crunching Information of Everything, KST 2017, Chonburi, Thailand, 1–4 February 2017*; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2017; pp. 254–259.
19. Yi, X.; Liu, F.; Niu, D.; Jin, H.; Lui, J.C.S. Cocoa: Dynamic Container-Based Group Buying Strategies for Cloud Computing. *ACM Trans. Model. Perform. Eval. Comput. Syst.* **2017**, *2*, 1–31. [CrossRef]
20. Khan, A.A.; Zakarya, M.; Khan, R.; Rahman, I.U.; Khan, M.; Khan, A.; Ur Rahman, I. An Energy, Performance Efficient Resource Consolidation Scheme for Heterogeneous Cloud Datacenters. *J. Netw. Comput. Appl.* **2020**, *150*, 102497. [CrossRef]
21. Hu, Y.; Zhou, H.; de Laat, C.; Zhao, Z. Concurrent Container Scheduling on Heterogeneous Clusters with Multi-Resource Constraints. *Future Gener. Comput. Syst.* **2020**, *102*, 562–573. [CrossRef]
22. Chung, A.; Park, J.W.; Ganger, G.R. Stratus: Cost-Aware Container Scheduling in the Public Cloud. In Proceedings of the 2018 ACM Symposium on Cloud Computing, Carlsbad, CA, USA, 11–13 October 2018; pp. 121–134.
23. Tumanov, A.; Jiang, A.; Woo Park, J.; Kozuch, M.A.; Ganger, G.R. *JamaisVu: Robust Scheduling with Auto-Estimated Job Runtimes*; Technical Report CMU-PDL-16-104; Carnegie Mellon University: Pittsburgh, PA, USA, 2016; pp. 1–24.
24. Visti, H.; Kiss, T.; Terstyanszky, G.; Gesmier, G.; Winter, S. MiCADO-Towards a Microservice-Based Cloud Application-Level Dynamic Orchestrator. In Proceedings of the 8th International Workshop on Science Gateways, IWSG 2016, CEUR Workshop Proceedings, Rome, Italy, 8–19 June 2017; pp. 1–7.
25. Kiss, T.; Kacsuk, P.; Kovacs, J.; Rakoczi, B.; Hajnal, A.; Farkas, A.; Gesmier, G.; Terstyanszky, G. MiCADO—Microservice-Based Cloud Application-Level Dynamic Orchestrator. *Future Gener. Comput. Syst.* **2019**, *94*, 937–946. [CrossRef]
26. Kiss, T.; DesLauriers, J.; Gesmier, G.; Terstyanszky, G.; Pierantoni, G.; Oun, O.A.; Taylor, S.J.E.; Anagnostou, A.; Kovacs, J. A Cloud-Agnostic Queuing System to Support the Implementation of Deadline-Based Application Execution Policies. *Future Gener. Comput. Syst.* **2019**, *101*, 99–111. [CrossRef]
27. Kovács, J. Supporting Programmable Autoscaling Rules for Containers and Virtual Machines on Clouds. *J. Grid Comput.* **2019**, *17*, 813–829. [CrossRef]
28. Nadgowda, S.; Suneja, S.; Bila, N.; Isci, C. Voyager: Complete Container State Migration. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017; pp. 2137–2142.
29. Ding, Z.; Tian, Y.C.; Tang, M.; Li, Y.; Wang, Y.G.; Zhou, C. Profile-Guided Three-Phase Virtual Resource Management for Energy Efficiency of Data Centers. *IEEE Trans. Ind. Electron.* **2020**, *67*, 2460–2468. [CrossRef]
30. Zhou, R.; Li, Z.; Wu, C. An Efficient Online Placement Scheme for Cloud Container Clusters. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 1046–1058. [CrossRef]

31. Tan, B.; Ma, H.; Mei, Y. Novel Genetic Algorithm with Dual Chromosome Representation for Resource Allocation in Container-Based Clouds. In Proceedings of the IEEE International Conference on Cloud Computing, CLOUD, Milan, Italy, 8–13 July 2019; Volume 2019, pp. 452–456.
32. Boxiong, T.; Hui, M.; Yi, M. A NSGA-II-Based Approach for Service Resource Allocation in Cloud. In Proceedings of the 2017 IEEE Congress on Evolutionary Computation (CEC), San Sebastian, Spain, 5–8 June 2017; pp. 2574–2581.
33. Rossi, F.; Nardelli, M.; Cardellini, V. Horizontal and Vertical Scaling of Container-Based Applications Using Reinforcement Learning. In Proceedings of the 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), Milan, Italy, 8–13 June 2019; Volume 2019, pp. 329–338.
34. With Help from AI, Microservices Divvy up Tasks to Improve Cloud Apps | Cornell Chronicle. Available online: https://news.cornell.edu/stories/2019/03/help-ai-microservices-divvy-tasks-improve-cloud-apps (accessed on 14 March 2022).
35. Liu, H.; Zhang, J.; Shan, H.; Li, M.; Chen, Y.; He, X.; Li, X. JCallGraph: Tracing Microservices in Very Large Scale Container Cloud Platforms. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer: Cham, Switzerland, 2019; Volume 11513, pp. 287–302.
36. Dang-Quang, N.-M.; Yoo, M. An Efficient Multivariate Autoscaling Framework Using Bi-LSTM for Cloud Computing. *Appl. Sci.* **2022**, *12*, 3523. [CrossRef]
37. Malik, S.; Tahir, M.; Sardaraz, M.; Alourani, A. A Resource Utilization Prediction Model for Cloud Data Centers Using Evolutionary Algorithms and Machine Learning Techniques. *Appl. Sci.* **2022**, *12*, 2160. [CrossRef]
38. Malik, N.; Sardaraz, M.; Tahir, M.; Shah, B.; Ali, G.; Moreira, F. Energy-Efficient Load Balancing Algorithm for Workflow Scheduling in Cloud Data Centers Using Queuing and Thresholds. *Appl. Sci.* **2021**, *11*, 5849. [CrossRef]
39. Calderón-Gómez, H.; Mendoza-Pittí, L.; Vargas-Lombardo, M.; Gómez-Pulido, J.M.; Rodríguez-Puyol, D.; Sención, G.; Polo-Luque, M.-L. Evaluating Service-Oriented and Microservice Architecture Patterns to Deploy EHealth Applications in Cloud Computing Environment. *Appl. Sci.* **2021**, *11*, 4350. [CrossRef]
40. Górski, T.; Woźniak, A.P. Optimization of Business Process Execution in Services Architecture: A Systematic Literature Review. *IEEE Access* **2021**, *9*, 111833–111852. [CrossRef]
41. Toosi, A.N.; Calheiros, R.N.; Buyya, R. Interconnected Cloud Computing Environments: Challenges, Taxonomy, and Survey. *ACM Comput. Surv.* **2014**, *47*, 1–47. [CrossRef]
42. Hemmat, R.A.; Hafid, A. SLA Violation Prediction in Cloud Computing: A Machine Learning Perspective. *arXiv* **2016**, arXiv:1611.10338. [CrossRef]
43. Vasudevan, M.; Tian, Y.-C.; Tang, M.; Kozan, E.; Zhang, X. Energy-Efficient Application Assignment in Profile-Based Data Center Management through a Repairing Genetic Algorithm. *Appl. Soft Comput.* **2018**, *67*, 399–408. [CrossRef]
44. Yang, H.; Breslow, A.; Mars, J.; Tang, L. Bubble-Flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers. *ACM SIGARCH Comput. Archit. News* **2013**, *41*, 607–618. [CrossRef]
45. Qureshi, B. Profile-Based Power-Aware Workflow Scheduling Framework for Energy-Efficient Data Centers. *Future Gener. Comput. Syst.* **2019**, *94*, 453–467. [CrossRef]
46. Yu, Y.; Yang, J.; Guo, C.; Zheng, H.; He, J. Joint Optimization of Service Request Routing and Instance Placement in the Microservice System. *J. Netw. Comput. Appl.* **2019**, *147*, 102441. [CrossRef]
47. Mekala, M.S.; Viswanathan, P. Energy-Efficient Virtual Machine Selection Based on Resource Ranking and Utilization Factor Approach in Cloud Computing for IoT. *Comput. Electr. Eng.* **2019**, *73*, 227–244. [CrossRef]
48. Rossi, F.; Cardellini, V.; Lo Presti, F.; Nardelli, M. Geo-Distributed Efficient Deployment of Containers with Kubernetes. *Comput. Commun.* **2020**, *159*, 161–174. [CrossRef]
49. Chen, F.; Grundy, J.; Yang, Y.; Schneider, J.-G.; He, Q. Experimental Analysis of Task-Based Energy Consumption in Cloud Computing Systems. In Proceedings of the ACM/SPEC International Conference on International Conference on Performance Engineering—ICPE '13, Prague, Czech Republic, 21–24 April 2013; ACM Press: New York, NY, USA, 2013; pp. 295–306.
50. Chen, F.; Grundy, J.; Schneider, J.-G.; Yang, Y.; He, Q. Automated Analysis of Performance and Energy Consumption for Cloud Applications. In Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering, Dublin, Ireland, 22–26 March 2014; ACM: New York, NY, USA, 2014; pp. 39–50.
51. Cambronero, M.E.; Bernal, A.; Valero, V.; Cañizares, P.C.; Núñez, A. Profiling SLAs for Cloud System Infrastructures and User Interactions. *PeerJ Comput. Sci.* **2021**, *7*, e513. [CrossRef] [PubMed]
52. Liu, Z.; Yu, H.; Fan, G.; Chen, L. Reliability Modelling and Optimization for Microservice-based Cloud Application Using Multi-agent System. *IET Commun.* **2022**, 1–18. [CrossRef]
53. Khan, A. Key Characteristics of a Container Orchestration Platform to Enable a Modern Application. *IEEE Cloud Comput.* **2017**, *4*, 42–48. [CrossRef]
54. Wolke, A.; Bichler, M.; Setzer, T. Planning vs. Dynamic Control: Resource Allocation in Corporate Clouds. *IEEE Trans. Cloud Comput.* **2016**, *4*, 322–335. [CrossRef]

55. Morikawa, T.; Kourai, K. Low-Cost and Fast Failure Recovery Using In-VM Containers in Clouds. In Proceedings of the 2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), Fukuoka, Japan, 5–8 August 2019; pp. 572–579.
56. Assuncao, W.K.G.; Colanzi, T.E.; Carvalho, L.; Pereira, J.A.; Garcia, A.; de Lima, M.J.; Lucena, C. A Multi-Criteria Strategy for Redesigning Legacy Features as Microservices: An Industrial Case Study. In Proceedings of the 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), Honolulu, HI, USA, 9–12 March 2021; pp. 377–387.
57. Saboor, A.; Mahmood, A.K.; Hassan, M.F.; Shah, S.N.M.; Hassan, F.; Siddiqui, M.A. Design Pattern Based Distribution of Microservices in Cloud Computing Environment. In Proceedings of the International Conference on Computer & Information Sciences (ICCOINS), Kuching, Malaysia, 13–15 July 2021; pp. 396–400.
58. Saboor, A.; Mahmood, A.K.; Omar, A.H.; Hassan, M.F.; Shah, S.N.M.; Ahmadian, A. Enabling Rank-Based Distribution of Microservices among Containers for Green Cloud Computing Environment. *Peer-to-Peer Netw. Appl.* **2022**, *15*, 77–91. [CrossRef]
59. Piraghaj, S.F.; Dastjerdi, A.V.; Calheiros, R.N.; Buyya, R. ContainerCloudSim: An Environment for Modeling and Simulation of Containers in Cloud Data Centers. *Softw. Pract. Exp.* **2017**, *47*, 505–521. [CrossRef]
60. Gan, Y.; Zhang, Y.; Cheng, D.; Shetty, A.; Rathi, P.; Katarki, N.; Bruno, A.; Hu, J.; Ritchken, B.; Jackson, B.; et al. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, Providence, RI, USA, 13–17 April 2019; ACM: New York, NY, USA, 2019; pp. 3–18.